Concurrent Multi-Target Tracking

T. D. Gottschalk
California Institute of Technology
Pasadena, CA 91125

1 Overview

Simulation89 is an emulation of various SDI tasks (tracking, engagement management and 'look ahead') developed for the U. S. Air Force. The simulation presently deals with the boost, post-boost and early midcourse phases of a 'mass raid' scenario, and is designed to process scenarios with a few thousand targets. The simulation is run on the Mark-III hypercube, with individual tasks performed on subcubes of the full hypercube. In general, the computations within individual subcubes are done in a synchronous manner (i.e., *CrOS*), while communications between tasks/subcubes are done asynchronously.

The nominal task for the tracking module is to provide state information on individual targets, given 2D line of sight data from various sensors at regular time intervals. This task is complicated by means of a few relevant additional requirements:

1. In the initial boost phase, the trajectories of individual targets are not fully known.
2. The overall system must scale in such a way that increases in the size of the underlying scenario are accomodated by (proportional) increases in the size of the tracking sub-cube.
3. The tracker must meet 'real time' requirements.

The first requirement in fact dictates the gross overall structure of the tracking package, as illustrated in Fig.(1). A single tracking system is formed from two elementary 2D tracking subsystems. Each 2D tracking sub-system processes individual data from its own associated sensor, forming lists of plausible mono tracks through these data sets. These 2D tracks are then shared between the two 2D sub-systems, and a single set of 3D tracks is formed.

The tracking models used for the 2D and 3D subsystems are quite different. According to the first requirement listed above, it must be assumed that the data from a single sensor are *insufficient* to resolve all Track↔Hit ambiguities. As a consequence, the 2D systems use a *Multiple Hypothesis* formalism in which

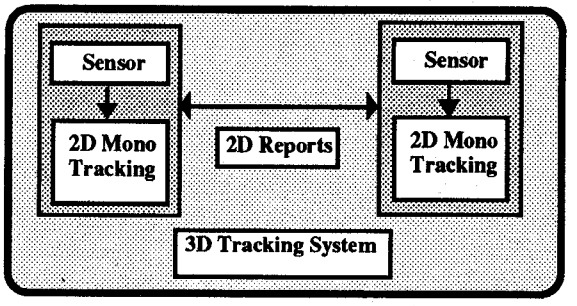

Figure 1: Schematic Tracker Organization

many candidate tracks through a single sensor datum are allowed. Such a model is subject to exponential explosions of the overall track file. This fundamental difficulty is resolved by a number of rules for pruning the size of the overall track file. In particular

1. Two tracks ending on a given datum are said to be equivalent if they share the same 2D data over the last four scans.
2. The number of inequivalent tracks per datum is limited by a cutoff parameter.
3. The total number of 2D tracks is also limited by a global cutoff.

If two tracks in the system are found to be equivalent according to point 1, one of the tracks is simply deleted. As is discussed below, the task of identifying equivalent tracks in the distributed 2D track file dictates the maner in which the 2D tracking problem is decomposed for concurrent execution.

Unlike the 2D tracking system, the 3D tracker in Fig.(1) maintains (at most) one track per sensor data point, representing the best global interpretation of tracks through the data (this single 'best guess' answer is the output of the tracker expected by the other elements of Sim89). In place of the Multiple-Hypothesis model used for 2D tracking, the

0-8186-2113-3/90/0000/0085$01.00 © 1990 IEEE

3D tracker is based on *Optimal Associations*. These optimal associations in fact come in two distinct forms:

1. For track extensions, the predicted data positions of individual 3D tracks are associated with actual data from the two sensing subsystems of Fig.(1).
2. For 3D track initiations, 2D tracks form the two subsystems of Fig.(1) are associated according to values of projections onto an association reference axis (so-called 'Hinge Angle' associations).

The adoption of an Optimal Association formalism essentially trivializes the concurrent decomposition of the 3D tracker : the 3D tracks are distributed among the nodes of the subcube in such a way that the number of tracks per node is constant. The challenge of concurrent 3D tracking comes entirely in performing the two types of optimal associations.

A general concurrent algorithm for optimal associations is described in Ref.[1]. However, the resource requirements for the general optimal association problem ($\Delta T \propto N^3$ for $N \times N$ association problems) is such that a straightforward use of the general association formalism is completely inappropriate. Instead, the concurrent association algorithm proceeds as follows:

1. Each node computes a list of associations keys (i.e., projections onto an appropriate reference axis), for all items in its local track list.
2. The distributed lists of association keys are globally sorted.
3. The sorted lists are divided into a number of sub-blocks, determined by appropriately large gaps in the lists of keys.
4. The sub-blocks are assigned to individual nodes and the assignment problems for sub-blocks are solved using a modified 'sparse' formalism of the general assignment problem.

This procedure is efficient as long as the number of separate sub-blocks found in the third step is large compared to the number of nodes in the tracking subcubes (which is, empirically, almost always the case for the Sim89 problem).

In addition to the central tasks of Track↔Hit and Track↔Track associations, the 3D tracker also evaluates trajectory fits for all 3D tracks in the system. Unlike the predecessors to Sim89, these trajectory fits are not essential elements of tracking *per se*. All tracking is done using kinematic system models. The trajectory parameterizations are added to the tracking task

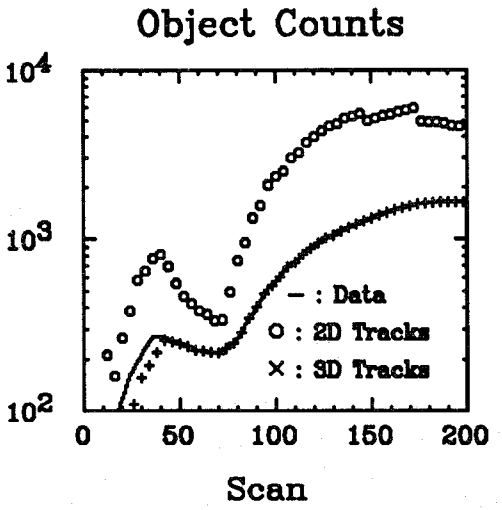

Figure 2: Data Set and Track File Sizes Versus Scan

for purposes of communications: The 3D track file structures according to the kinematic model are huge (more than 100 floating point numbers per track), and the model-dependent parameterizations greatly reduce the sizes of track file messages passed between subcubes of the full Sim89 simulation. The concurrent estimation of track parameters is again trivial, with each node independently performing this task for its own subset of the global track file.

2 Some Sample Results

This section briefly examines some typical results of the Sim89 tracker for a standard input set. The threat scenario involves 200 primary targets, each of which ultimately spawns 10 daughter objects (RV's). The targets are launched from six separated launch sites over a two minute time window. The primary threat is preceeded by a simultaneous launch of sixty secondary targets (ASAT's).

Sizes of the data sets and 2D and 3D track files are plotted versus scan number in Fig.(2). The peaks near scan 40 are due to interception of the ASAT's, while the prolonged increase in object counts after scan 80 is due to gradual deployment of RV's from the surviving primary targets. As expected, the number of 2D tracks greatly exceeds the actual number of targets. The 'kinks' in the 2D track counts for large scan number are the result of the automatic reductions in tracks/datum cutoffs mentioned in Section 1.

The number of 3D tracks is very close to the actual number of targets. The histogram in Fig.(3) shows

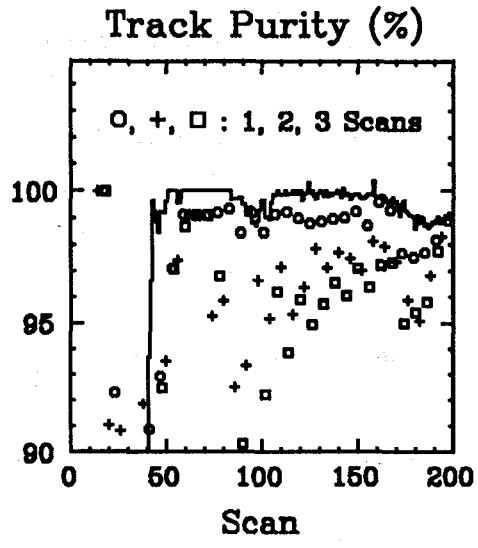

Figure 3: Track File Completeness, Purity Versus Scan

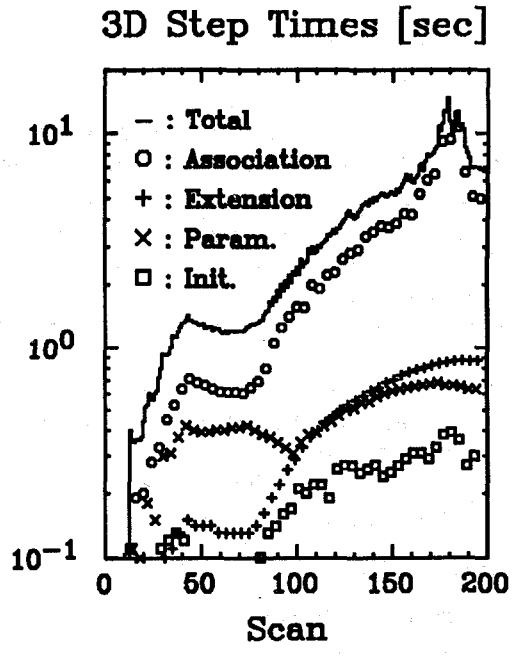

Figure 4: CPU Step Times For 3D Tracking

the percentage of targets in track,

$$P[\text{In Track}] \equiv N[\text{3D Tracks}]/N[\text{Data}]$$

versus scan number. Once the primary targets are into second stage (about scan 50), the percentage in track is excellent. Also shown in Fig.(3) are the fractions of pure tracks

$$P_j \equiv N[\text{j-Scans Correct}]/N[\text{3D Tracks}]$$

where the numerator is the number of 3D tracks which (correctly) incorporate data from a single underlying target through the past j scans.

The mild degradations in both percentage in track and j-Scan correct tracks between scans 150 and 200 are due to the successes of the engagement management component of Sim89 in intercepting the targets. The disappearence of expected targets causes some 'confusion' for Track↔Hit associations on subsequent tracking scans.

The CPU times for various components of 3D tracking are plotted versus scan number in Fig.(4). Most of the CPU resources are spent in the evaluation of Track↔Hit associations. With the exception of the 'confused' scans with disapperaing tracks, the CPU requirements for tracking generally scale as $\Delta T \propto NlogN$ for N active targets.

3 Concurrent Aspects

The task of multi-target tracking is well-suited for concurrent execution, with most of the 'tracking' *per se* done by way of CPU-intensive operations involving individual track↔data pairs (the filter update of a single 3D track involes more than one thousand floating point operations : an ideal use of the WEITEK coprocessor). In the entire tracking program (more than 35000 lines of code), there are really only three general concurrent operations/aspects,

1. Global collection of data across the subcube.
2. Distributed sorting.
3. Track file redistributions.

with each of these tasks occuring in a variety of guises. The sorting task is done using the basic algorithm of Ref.[2], with a trivial but important modification to allow empty local sublists (empty track files on some nodes occur during the first few scans of the tracking task).

The global data collections are all done using a simple loop on communication channels:

- `Set Global Value To Local Value`
- `Loop On Communication Channels`
  - `Exchange Values Across Channel`

- **Update Global Value Using Input**

- **End Of Channel Loop**

This simple paradigm is used throughout the code for a variety of purposes, such as assessment of global status (the 'Update' task is a logical and of individual status flags), determining global file sizes ('Update' is simple addition) or assessing global Track↔Data assignments ('Update' is a slightly more complicated merging of track assignment arrays generated on individual nodes).

For the 3D tracking task, concurrent efficiency requires only that the number of tracks per node be approximately constant. Accordingly, track file redistribution for 3D tracking is done using a simple variant of the channel loop model:

- **Loop On Communication Channels**
  - **Exchange Track File Sizes**
  - **Set** $\delta \equiv (N_{\mathtt{HERE}} - N_{\mathtt{THERE}})/2$
  - **If** $\delta > 0$**, Send** δ **Items Over Channel.**
  - **If** $\delta < 0$**, Receive** δ **Items.**
- **End Of Loop On Channels.**

After the exchanges across a given channel, the number of items on each half of the subcube with respect to that channel is (approximately) the same, and subsequent loops on other channels do not modify this equality. At the end of the channel loop, the tracks are equally divided across *all* channels - meaning that the number of tracks per node must be approximately the same.

The only aspect of the full tracking program which involves concurrent 'subtleties' is the redistribution of the the global 2D track files. Wasteful cube-wide searches for equivalent tracks (same sensor data over the last four scans) can be avoided if the assignment of tracks to nodes is done according to a single essential requirement.

> At the start of each 2D tracking scan, all tracks ending on a given sensor datum are to be assigned to a *single* node.

If this condition is satisfied, then searches for equivalent tracks need only be done locally. The requirement is enforced as follows:

1. Assign each datum of the current data set to a specific node.
2. Transfer all tracks in the system to that node which 'owns' the data point for the last scan included in the track.

This redistribution is in fact done as the last step in 2D processing at each scan, so that the next scan begins with the basic track distribution requirement satisfied.

Once data points (hence tracks) have been assigned to individual nodes, the actual redistribution of the tracks is a straightforward application of the basic *Crystal_Router* formalism of Ref.[3]. The calculation of destinations for individual data is done using the following simple set of rules:

1. Each datum is assigned a *Weight*, taken to be the total number of tracks in the system wich end at that datum.
2. Data are assigned to nodes such that the total Weight per node is approximately constant.
3. If, prior to the redistribution, a particular node already contains more than half of the total weight of an individual datum, then the datum is assigned to that node - provided that such an assignment does not violate the total node weight restrictions of point 2.
4. Unassigned data (i.e., data without tracks) are assigned to nodes in a simple 'Card Dealing' fashion.

These rules are easily implemented by means of a few simple channel loops of the form discussed above.

4 Conclusion

The hard part of 'Concurrent Tracking' is the tracking iteslf, *not* the concurrency.

References

1. T. D. Gottschalk, 'Concurrent Implementation of Munkres Algorithm', there proceedings.
2. G. C. Fox *et al., Solving Problems on Concurrent Processors,* Englewood Cliffs, NJ: Prentice Hall(1988), Chapter 18.
3. Chapter 22 of Ref.[2].